OIPE JCe
JUL 0 6 2004
PATENT & TRADEMARK OFFICE

The datestamp of the U.S. Patent and Trademark Office hereon will acknowledge receipt of the following item(s):					
Docket No.	2207/17040	*Serial No.*	TBA	*Date of Mailing*	December 30, 2003
By	ESQ/PMR/SJO	*Filing Date*	TBA	*Express Mail No.*	EV351180931US
Inventors/ Applicant	Avinash SODANI et al.				
Title	METHOD AND APPARATUS FOR RESCHEDULING OPERATIONS IN A PROCESSOR				

Item No.	*Description*	*No. of Pages*
1.	UTILITY PATENT APPLICATION TRANSMITTAL (plus 1 copy)	2
2.	UTILITY PATENT APPLICATION	18
3.	FORMAL DRAWINGS	7
4.	DECLARATION AND POWER OF ATTORNEY (unsigned)	3
5.		
6.		

19270 U.S. PTO
~~10/749272~~
123003

EV 351180931 US

EXPRESS MAIL
UNITED STATES POSTAL SERVICE®
Post Office

4/6/

ORIGIN (POSTAL USE ONLY)			
PO ZIP Code 95113	Day of Delivery [ ] Next [✓ Second]	Flat Rate Envelope [ ]	
Date In Mo. 12 Day 30 Year 03	[ ] 12 Noon [ ] 3 PM	Postage $ 13.65	
Time In 429 [✓ AM] [ ] PM	Military [ ] 2nd Day [ ] 3rd Day	Return Receipt Fee	
Weight 6 lbs. 1 ozs.	Int'l Alpha Country Code	COD Fee	Insurance Fee
Delivery [ ] Weekend [ ] Holiday	Acceptance Clerk Initials	Total Postage & Fees $ 13.65	

St James Park Sta
95113-9998
H10
DEC 30 2003

DELIVERY (POSTAL USE ONLY)		
Delivery Attempt Mo. Day	Time [ ] AM [ ] PM	Employee Signature
Delivery Attempt Mo. Day	Time [ ] AM [ ] PM	Employee Signature
Delivery Date Mo. Day	Time [ ] AM [ ] PM	Employee Signature

WAIVER OF SIGNATURE (Domestic Only) Additional merchandise insurance is void if waiver of signature is requested. I wish delivery to be made without obtaining signature of addressee or addressee's agent (if delivery employee judges that article can be left in secure location) and I authorize that delivery employee's signature constitutes valid proof of delivery.

NO DELIVERY [ ] Weekend [ ] Holiday

Customer Signature

Service Guarantee
If the shipment ... to the address ... attempt is no... enforcement ... government ... network ... A notice ... within ... Please ... Manu...

CUSTOMER USE ONLY
METHOD OF PAYMENT:
Express Mail Corporate Acct. No.

Federal Agency Acct. No. or Postal Service Acct. No.

FROM: (PLEASE PRINT) PHONE 408 975 7500

KENYON & KENYON
333 W SAN CARLOS ST STE 600
SAN JOSE CA 95110-2731

ESQ/PMR/SJO 2207/17040

TO: (PLEASE PRINT) PHONE ( )

MAIL STOP PATENT APPLICATION
COMMISSIONER FOR PATENTS
PO BOX 1450
ALEXANDRIA VA 22313-1450

PRESS HARD. You are making 3 copies.

FOR PICKUP OR TRACKING CALL 1-800-222-1811 www.usps.com

EMS

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

UTILITY PATENT APPLICATION TRANSMITTAL UNDER 37 CFR 1.53(b)

Address to: M/S: PATENT APPLICATION Commissioner for Patents P.O. Box 1450 Alexandria, VA 22313-1450	***Attorney Docket No.***	2207/17040
	Inventor(s)	Avinash SODANI et al.
	Express Mail Label No.	EV351180931US
	Total Pages	30

Title of Application:

METHOD AND APPARATUS FOR RESCHEDULING OPERATIONS IN A PROCESSOR

Transmitted with the patent application are the following:

2	Page(s)	Transmittal form (and one copy)
18	Page(s)	Cover Page (1); Specification (11) Claims (5), Abstract (1)
7	Page(s)	Formal drawings
3	Page(s)	Declaration and Power of Attorney (unsigned)
____	Page(s)	Recordation of Assignment and Assignment form
____	Page(s)	Information Disclosure Statement (IDS) (copies of citations not included in number of pages)
____	Page(s)	Certified copy of:
____	Page(s)	Statement of Related Applications

This application is a ___ Continuation / Continuation-in-Part / Divisional of prior application Serial No. _______, filed ______.

Fee calculation for large entity:	***No. Filed***	***Allowed***	***No. Extra***	***Rate***	***Fee***
Basic Fee					$770.00
Total Claims	24	-20	4	18.00	$72.00
Independent Claims	4	-3	1	86.00	$86.00
Multiple Dependent Claim				+$290.00	
				Assignment	
				Total	$928.00

The Commissioner is hereby authorized to charge payment of any fees associated with this communication or arising during the pendency of this application per 37 CFR §1.16-1.21, or to credit any overpayments, to Deposit Account 11-0600.

Customer Number: **25693**

Express Mail Certificate

I hereby certify that the above paper/fee is being deposited with the United States Postal Service Express Mail Post Office to Addressee service under 37 CFR 1.10 on the date indicated below and is addressed to the M/S: PATENT APPLICATION, P. O. Box 1450, Alexandria, VA 22313-1450

Date: December 30, 2003 Signature Pilar Rodriguez
Pilar Rodriguez

Respectfully submitted,

[illegible]
Edward S. Quon (Reg. No. 52,144)
Attorney for Applicant(s)

Date: December 30, 2003

2207/17040

PATENT

UNITED STATES PATENT APPLICATION
FOR

METHOD AND APPARATUS FOR RESCHEDULING OPERATIONS IN A PROCESSOR

INVENTORS:

AVINASH SODANI
PER H. HAMMARLUND
STEPHAN J. JOURDAN

PREPARED BY:

KENYON & KENYON
333 WEST SAN CARLOS STREET, SUITE 600
SAN JOSE, CALIFORNIA 95110
(408) 975-7500

Ex. Mail No. EV351180931US
46542.1

METHOD AND APPARATUS FOR RESCHEDULING OPERATIONS IN A PROCESSOR

Background of the Invention

[0001] The primary function of most computer processors is to execute computer instructions. Most processors execute instructions in the programmed order that they are received. However, some recent processors, such as the Pentium®. II processor from Intel Corp., are "out-of-order" processors. An out-of-order processor can execute instructions in any order as the data and execution units required for each instruction becomes available. Therefore, with an out-of-order processor, execution units within the processor that otherwise may be idle can be more efficiently utilized.

[0002] With either type of processor, delays can occur when executing "dependent" instructions. A dependent instruction, in order to execute correctly, requires a value produced by another instruction that has executed correctly. For example, consider the following set of instructions:

1) Load memory-1 into register-X;

2) Add1 register-X register-Y into register-Z;

3) Add2 register-Y register-Z into register-W.

[0003] The first instruction loads the content of memory-1 into register-X. The second instruction adds the content of register-X to the content of register-Y and stores the result in register-Z. The third instruction adds the content of register-Y to the content of register-Z and stores the result in register-W. In this set of instructions, instructions 2 and 3 are dependent instructions that are dependent on instruction 1 (instruction 3 is also dependent on instruction 2). In other words, if register-X is not loaded with the proper value in instruction 1 before instructions 2 and 3 are executed, instructions 2 and 3 will likely generate incorrect results.

Dependent instructions can cause a delay in known processors because most known processors typically do not schedule a dependent instruction until they know that the instruction that the dependent instruction depends on will produce the correct result.

[0004] Referring now to the drawings, FIG. 1 is a block diagram of a processor pipeline and timing diagram illustrating the delay caused by dependent instructions in most known processors. In FIG. 1, a scheduler 105 schedules instructions. The instructions proceed through an execution unit pipeline that includes pipeline stages 110, 115, 120, 125, 130, 135 and 140. During each pipeline stage a processing step is executed. For example, at pipeline stage 110 the instruction is dispatched. At stage 115 the instruction is decoded and source registers are read. At stage 120 a memory address is generated (for a memory instruction) or an arithmetic logic unit ("ALU") operation is executed (for an arithmetic or logic instruction). At stage 125 cache data is read and a lookup of the translation lookaside buffer ("TLB") is performed. At stage 130 the cache Tag is read. At stage 135 a hit/miss signal is generated as a result of the Tag read. The hit/miss signal indicates whether the desired data was found in the cache (i.e., whether the data read from the cache at stage 125 was the correct data). As shown in FIG. 1, the hit/miss signal is typically generated after the data is read at stage 125, because generating the hit/miss signal requires the additional steps of TLB lookup and Tag read.

[0005] The timing diagram of FIG. 1 illustrates the pipeline flow of two instructions: a memory load instruction ("Ld") and an add instruction ("Add"). The memory load instruction is a six-cycle instruction, the add instruction is a one-cycle instruction, and the add instruction is dependent on the load instruction. At time=0 (i.e., the first clock cycle) Ld is scheduled and dispatched (pipeline stage 110). At time=1, time=2 and time=3, Ld moves to pipeline stages 115, 120 and 258, respectively. At time=4, Ld is at pipeline stage 130. At time=5, Ld is at stage 135 and the hit/miss signal is generated. Scheduler 105 receives this signal. Finally at time=6, assuming a hit signal is received indicating that the data was correct, scheduler 105 schedules Add to stage 110, while Ld continues to stage 140, which is an additional pipeline stage. The add operation is eventually performed when Add is at stage 120. However, if at time=6 a miss signal is received, scheduler 105 will wait an indefinite number of clock cycles until data is received by accessing the next levels of the cache hierarchy.

[0006] As shown in the timing diagram of FIG. 1, Add, because it is dependent on Ld, cannot be scheduled until time=6, at the earliest. A latency of an instruction may be defined as the time from when its input operands must be ready for it to execute until its result is ready to be used by another instruction. Therefore, the latency of Ld in the example of FIG. 1 is six. Further, as shown in FIG. 1, scheduler 105 cannot schedule Add until it receives the hit/miss signal. Therefore, even if the time required to read data from a cache decreases with improved cache technology, the latency of Ld will remain at six because it is dependent on the hit/miss signal.

[0007] Reducing the latencies of instructions in a processor is sometimes necessary to increase the operating speed of a processor. For example, suppose that a part of a program contains a sequence of N instructions, $I_1, I_2, I_3 \ldots I_N$. Suppose that I_{n+1} requires, as part of its inputs, the result of I_n, for all n, from 1 to N-1. This part of the program may also contain any other instructions. The program cannot be executed in less time than $T=L_1 +L_2 +L_3 + \ldots +L_N$, where L_n is the latency of instruction I_n, for all n from 1 to N. In fact, even if the processor was capable of executing a very large number of instructions in parallel, T remains a lower bound for the time to execute this part of this program. Hence to execute this program faster, it will ultimately be essential to shorten the latencies of the instructions.

[0008] Based on the foregoing, there is a need for a computer processor that can schedule instructions, especially dependent instructions, faster than known processors, and therefore reduces the latencies of the instructions.

Brief Description of the Drawings

[0009] FIG. 1 is a block diagram of a prior art processor pipeline and timing diagram illustrating the delay caused by dependent instructions in most known processors.

[0010] FIG. 2 is a block diagram of a processor pipeline and timing diagram in accordance with one embodiment of the present invention.

[0011] FIG. 3 is a block diagram of a processor in accordance with one embodiment of the present invention.

[0012] FIG. 4 is a block diagram of a history-based re-scheduler in accordance with one embodiment of the present invention.

[0013] FIG. 5 is a flow diagram of the operation of a history-based re-scheduler in accordance with one embodiment of the present invention.

Detailed Description of the Drawings

[0014] One embodiment of the present invention is a processor that speculatively schedules instructions and that includes a replay system. The replay system replays instructions that were not executed correctly when they were initially dispatched to an execution unit. Further, the replay system preserves the originally scheduled order of the instructions.

[0015] FIG. 2 is a block diagram of a processor pipeline and timing diagram in accordance with one embodiment of the present invention. In FIG. 2, a scheduler 205 schedules instructions to pipeline stages 210, 215, 220, 225, 230, 235 and 240, which are identical in function to the stages shown in FIG. 1. The timing diagram of FIG. 2 illustrates a two-cycle Ld followed by a one-cycle Add. Scheduler 205 speculatively schedules Add without waiting for a hit/miss signal from Ld. Therefore, Add is scheduled at time=2, so that a two stage distance from Ld is maintained because Ld is a two-cycle instruction. Add is eventually executed at time=4 when it arrives at stage 220, which is one cycle after Ld performs the cache read at stage 225.

[0016] By speculatively scheduling Add, scheduler 205 assumes that Ld will execute correctly (i.e., the correct data will be read from the cache at stage 18). A comparison of FIG. 2 with FIG. 1 illustrates the advantages of speculatively scheduling Add. Specifically, in FIG. 1, the Add instruction was not scheduled until time=6, thus Ld had a latency of six. In contrast, in FIG. 2 the Add instruction was scheduled at time=2, thus Ld had a latency of only two, or four less than the Ld in FIG. 1. Further, scheduler 205 in FIG. 2 has slots available to schedule additional instructions at time=3 through time=6, while scheduler 10 in FIG. 1 was able to only schedule one add instruction by time=6. Therefore, the present invention, by speculatively scheduling, reduces the latency of instructions and is able to schedule and process more instructions than the prior art.

[0017] However, embodiments of the present invention account for the situation when an instruction is speculatively scheduled assuming that it will be executed correctly, but eventually is not executed correctly (e.g., in the event of a cache miss). The present invention resolves this problem by having a replay unit. The replay unit replays all instructions that executed incorrectly.

[0018] The replay unit is an efficient way to allow the instructions to be executed again. As is known in the art, the instructions remain fixed at the same relative position to the instructions they depend on as created by the scheduler when replayed in the replay loop. However, in embodiments of the present invention, when the instructions are replayed, they are sent to the history-based re-scheduler, where the instructions that are not ready are allowed to be delayed when necessary, increasing the efficiency of instructions output by the scheduler and also increasing the overall performance of the processor.

[0019] In practical implementation, the scheduler is a costly resource based on its relative on-die size. Since schedulers are expensive to implement, its overall on-die size should be minimized while its overall efficiency dispatching instructions should be maximized. For example, an instruction that is read from the instruction queue and written into the scheduler will wait in the scheduler until it is ready to be dispatched. Effectively, an instruction that waits in the scheduler for an extended period of time will waste an entry in the scheduler that could be otherwise used by other instructions in the queue that are already ready for dispatch. In embodiments of the present invention, the history-based re-scheduler analyzes which instructions will be ready for dispatch by the scheduler. Thus, the re-scheduler writes them into the scheduler in an order that decreases the duration the instructions are in the scheduler waiting to be scheduled and increases the efficiency of the scheduler without increasing its on-die size.

[0020] FIG. 3 is a block diagram of a computer processor in accordance with one embodiment of the present invention. The processor 305 is included in a computer system (not shown). Processor 305 is coupled to other components of the computer, such as a memory device 307 through a system bus 309.

[0021] Processor 305 includes an instruction queue 310. Instruction queue 310 feeds instructions into history-based re-scheduler 315. In one embodiment, the instructions are "micro-operations." (also known as "Uops"). Micro-operations are generated by translating complex instructions into simple, fixed length instructions for ease of execution.

[0022] History-based re-scheduler analyzes the instructions. Each instruction carries additional information with it that specifies how long it is likely to wait in the scheduler before scheduling based on previous dispatches and executions of the instruction. The information is used to delay writing in the scheduler those instructions that are likely to wait for at least a few clock cycles when there are instructions in line behind those that can be written to the scheduler and dispatched immediately. Scheduler 320 dispatches an instruction received from re-scheduler 315 when the resources are available to execute the instruction and when sources needed by the instruction are indicated to be ready.

[0023] Scheduler 320 speculatively schedules instructions because the instructions are scheduled when a source is indicated to be ready. Scheduler 320 outputs the instructions to a replay multiplexer ("MUX") 325. The output of replay MUX 325 is coupled to an execution unit 330. Execution unit 330 executes received instructions. Execution unit 330 can be an arithmetic logic unit ("ALU"), a floating point unit, a memory unit, etc. Execution unit 330 is coupled to registers 335 which are the registers of processor 305. Execution unit 330 loads and stores data in registers 335 when executing instructions.

[0024] Processor 305 further includes a replay unit 340. Replay unit 340 replays instructions that were not executed correctly after they were scheduled by scheduler 320. Replay unit 340, like execution unit 330, receives instructions output from replay MUX 325. Instructions are staged through replay unit 340 in parallel to being staged through execution unit 330. The number of stages varies depending on the amount of staging desired in each execution channel.

[0025] If the instruction has executed correctly, instruction is declared "replay safe" and is forwarded to a retirement unit 345 where it is retired. Retiring instructions is beneficial to processor 305 because it frees up processor resources and allows additional instructions to start execution. If the instruction has not executed correctly, replay unit 340 replays or re-executes the instruction by sending the instruction to re-scheduler 315. In the alternative, replay unit 340 may send the instructions to replay MUX 325.

[0026] An instruction may execute incorrectly for many reasons. The most common reasons are a source dependency or an external replay condition. A source dependency can occur when an instruction source is dependent on the result of another instruction. Examples of an external replay condition include a cache miss, incorrect forwarding of data (e.g., from a store buffer to a load), hidden memory dependencies, a write back conflict, an unknown data/address, and serializing instructions.

[0027] Replay unit 340 may determine that an instruction should be replayed based on an external signal (not shown). Execution unit 330 sends a replay signal to replay unit 340. The replay signal indicates whether an instruction has executed correctly or not. If the instruction has not executed correctly, the instruction can be sent to re-scheduler 315 and written into scheduler 320 again for efficient dispatch to execution units 330.

[0028] In one embodiment, processor 305 is a multi-channel processor. Each channel includes all of the components shown in FIG. 3. However, the execution unit 330 for each channel will differ. For example, execution unit 330 for one channel will be a memory unit, execution unit 330 for another channel will be an arithmetic unit, etc. Each channel includes its own replay unit 340.

[0029] In one embodiment, processor 305 is a multi-threaded processor. In this embodiment, replay unit 340 causes some of the threads to be retired while others are replayed. Therefore, replay unit 340 allows execution unit 305 to be more efficiently used by many threads.

[0030] FIG. 4 is a block diagram of a history-based re-scheduler in accordance with one embodiment of the present invention. The instruction queue 405 is coupled to history-based re-scheduler 410. Instruction queue 405 forwards instructions into re-scheduler 410. Functionally, re-scheduler 410 acts as a sorter, by sorting out the instructions by the number of clock cycles to delay each instruction. Re-scheduler 410 also receives instructions 415 from replay unit (not shown)

[0031] Re-scheduler 410 includes a re-scheduler multiplexer 420. Re-scheduler MUX 420 analyzes multiple instructions from instruction queue 405. The re-scheduler MUX 420 sorts the instructions so that those instructions that are specified to sit in scheduler 435 for the shortest amount of time are written into scheduler 435 before other instructions, thereby enabling greater scheduler performance and overall processor efficiency.

[0032] In one embodiment of the invention, re-scheduler MUX 425 is coupled to a delay block 425 which stores information for each instruction. Delay block 425 keeps information on the latency of each instruction and a history of how long each instruction waits in the scheduler before it is dispatched. The information in delay block 425 relating to the latency and "clock-cycle wait" history is first written during scheduling and is updated every time the instruction is executed.

[0033] In one embodiment in accordance with the present invention, a history-based prediction scheme may be utilized where delay block 425 records the previous execution outcome of an instruction. For example, delay block 425 records how many clock cycles an instruction waits in scheduler 435, how long it takes to execute, and accounts for any resource conflicts that may increase the time to complete execution of the instruction. In FIG. 4, as shown, re-scheduler MUX 420 uses the information from delay block 425 to implement "delay lines" within delay queue 430 where instructions with a higher range of values in the wait field are delayed for a fixed number of clocks before they are written into the scheduler. Those instructions with a lower range of values in the wait field are delayed for a smaller fixed number of clocks before they are written into the scheduler. For example, delay line "0" may indicate that the instructions in this line should be written into the scheduler 435 immediately, whereas those sorted into delay line "3" should wait three clock cycles before being written to the scheduler 435.

[0034] In this embodiment of the invention, delay queue 430 includes delay line 0 through delay line 3. However, the number of delay lines in delay queue 430 may vary. The number of fixed clock cycles for each delay line may vary, as well. In this manner, re-scheduler MUX 420 can sort out various instructions, by placing them into individual "delay lines," and writing them into the scheduler after the fixed number of clocks have past. Thus, the scheduler is not clogged by instructions that cannot schedule with only a short delay, thereby resulting in a more efficient scheduler.

[0035] In one embodiment, the delay block 425 can predict the wait times based on a general prediction scheme. For example, when looking at the relative program position of two instructions, the latency of the first instruction may predict the wait time for a second dependent instruction. If they are close together in the program, but the first instruction has a long latency, then delay block 425 will predict that the second instruction should have a wait time at least the length of the latency of the first instruction. Other types of predictions schemes can also be implemented based on the type of instructions executed by the program.

[0036] FIG. 5 is a flow diagram of the history-based re-scheduler in accordance with one embodiment of the present invention. FIG. 5 illustrates the operation of the re-scheduler 410 according to one embodiment of the present invention. In block 505, the re-scheduler 410 analyzes incoming Uops from the instruction queue 405 or the replay unit, and control is then forwarded to block 510. In block 510, re-scheduler MUX 420 checks delay block 425 for latency and scheduling history information for each Uops being analyzed. Re-scheduling MUX 420 then determines, utilizing the scheduling history from delay block 425, the "wait" time for each Uop in block 515. Control passes to block 520, and re-scheduler MUX 420 places the Uops in the delay queue 430, and more specifically, determines the optimal delay line for each Uop based on the value of the "wait" time and range of "wait" values for the delay lines. In block 525, delay queue 430 writes the Uops from each delay line after the fixed clock cycles have passed. Thereby, re-scheduler 410 increases the overall efficiency of the processor by decreasing the latency of scheduler dispatches into execution.

[0037] While the description above refers to particular embodiments of the present invention, it will be understood that many modifications may be made without departing from the spirit thereof. The accompanying claims are intended to cover such modifications as would fall within the true scope and spirit of the present invention. The presently disclosed embodiments are therefore to be considered in all respects as illustrative and not restrictive, the scope of the invention being indicated by the appended claims, rather than the foregoing description, and all changes that come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

What is claimed is:

1. A re-scheduling system comprising:

a re-scheduler device coupled to an instruction queue to receive an instruction;

a delay unit coupled to said re-scheduling multiplexer to store a wait history for said instruction; and

a delay queue coupled to the output of said re-scheduler device to hold said instruction prior to writing said instruction into a scheduler.

2. The system of claim 1 wherein said delay queue comprises:

said at least one delay line, wherein each of said at least one delay line is to hold said instruction for a fixed number of clock cycles.

3. The system of claim 2 wherein said re-scheduler device is to sort said instruction into at least one delay line based on said wait history for said instruction.

4. The system of claim 1 wherein said delay unit is to store wait history information for said instruction when said instruction is first scheduled, and is to update wait history information for said instruction when said instruction is executed.

5. The system of claim 1 wherein said delay unit is to store latency information for said instruction when said instruction is first scheduled, and is to update latency information for said instruction when said instruction is executed.

6. The system of claim 1 wherein said delay unit is to store resource conflicts for said instruction when said instruction encounters a resource conflict during execution.

7. The system of claim 2 wherein said delay block utilizes a general prediction scheme to determine a number of clock cycles to store as said wait history.

8. The system of claim 7 wherein said re-scheduler device sorts said instruction based on said wait history for said instruction.

9. A computer processing system comprising:

a memory device to store instructions;

a processor coupled to said memory device and to execute said instructions, said processor including:

a re-scheduler having a first input coupled to an instruction queue, a second input coupled to a replay unit, and an output;

a scheduler coupled to said re-scheduler output;

a replay system having a first and second outputs, said first output coupled to an execution unit and second output coupled to said replay unit;

wherein said re-scheduler comprises:

a re-scheduler device coupled to an instruction queue to receive an instruction;

a delay unit coupled to said re-scheduler device to store wait history for said instruction; and a delay queue coupled to the output of said re-scheduler device to hold said instruction for a fixed number of clock cycles.

10. The system of claim 9 wherein said delay queue comprises:

said at least one delay line, wherein each of said at least one delay line holds said instruction for a fixed number of clock cycles.

11. The system of claim 10 wherein said re-scheduler device is to sort said instruction into at least one delay line based on said wait history for said instruction.

12. The system of claim 9 wherein said delay unit is to store wait history information for said instruction when said instruction is first scheduled, and is to update wait history information for said instruction when said instruction is executed.

13. The system of claim 9 wherein said delay unit is to store latency information for said instruction when said instruction is first scheduled, and is to update latency information for said instruction when said instruction is executed.

14. The system of claim 9 wherein said delay unit is to store resource conflicts for said instruction when said instruction encounters a resource conflict during execution.

15. The system of claim 10 wherein said delay unit utilizes a general prediction scheme to determine a number of clock cycles to store as said wait history.

16. The system of claim 15 wherein said re-scheduler device is to sort said instruction based on said wait history for said instruction.

17. A method of processing a computer instruction in a replay loop comprising:

receiving an instruction output from an instruction queue;

checking a delay unit for a scheduling history for said instruction;

determining a wait time for said instruction;

placing said instruction in a delay queue, wherein said delay queue includes at least one delay line with a fixed wait time; and

writing said instruction to said scheduler after said fixed wait time.

18. The method of claim 17 wherein determining a wait time for said instruction is based on said scheduling history stored in said delay unit.

19. The method of claim 18 wherein placing said instruction in a delay queue comprises:

matching said wait time for said instruction to the closest said fixed wait time for said at least one delay line.

20. The method of claim 17 wherein said scheduling history is determined by a general prediction scheme.

21. A set of instructions residing in a storage medium, said set of instructions capable of being executed by a processor to implement a method of processing a computer instruction in a replay loop comprising:

receiving an instruction output from an instruction queue;

checking a delay unit for a scheduling history for said instruction;

determining a wait time for said instruction;

placing said instruction in a delay queue, wherein said delay queue includes at least one delay line with a fixed wait time; and

writing said instruction to said scheduler after said fixed wait time.

22. The set of instructions of claim 21 wherein determining a wait time for said instruction is based on said scheduling history stored in said delay unit.

23. The set of instructions of claim 22 wherein placing said instruction in a delay queue comprises:

matching said wait time for said instruction to the closest said fixed wait time for said at least one delay line.

24. The set of instructions of claim 21 wherein said scheduling history is determined by a general prediction scheme.

Abstract of the Disclosure

A method and apparatus for rescheduling operations in a processor. More particularly, the present invention relates to optimally using a scheduler resource in a processor by analyzing, predicting, and sorting the write order of instructions into the scheduler so that the duration the instructions sit idle in the scheduler is minimized. The analyses, prediction, and sorting may be done between an instruction queue and a scheduler by using delay units. The prediction can be based on history (latency, dependency, and resource) or on a general prediction scheme.